evalctl: Local-first evals for agents

evalctl: Local-first evals for agents

Local-first evals for agents, not just prompts. evalctl treats eval cases as files, runners as shell commands, and results as durable artifacts. It scores what agents actually do — files written, diffs produced, commands run — on your own machine, with no gateway, dashboard, or SaaS account.

About

evalctl is an agent-first evaluation harness for local workflows. It runs eval cases as filesystem fixtures, invokes agents through ordinary runner commands, and grades the resulting workspace with deterministic scorers. The core artifact is a portable run directory that another agent can inspect, report on, and re-score without access to the original shell history.

Keywords: agent evals, LLM evaluation, AI agents, local-first, CLI, code review, workspace diff, deterministic scoring, artifact replay, eval harness.

How it differs from promptfoo

promptfoo is the incumbent local eval CLI, and it is prompt/chat-shaped. evalctl is agent-shaped.

- Unit under test: promptfoo scores a prompt to a completion; evalctl scores an agent run — files, diffs, command logs, artifacts.
- Scoring surface: promptfoo grades the text of a response; evalctl grades the resulting workspace — git diff, expected and forbidden file changes, exit codes, plus text.
- Execution: evalctl is synchronous by default; plan previews actions, doctor diagnoses state, run --resume resumes crashed runs; an optional queue can delegate runner execution.
- Model context: optional inferctl preflight provenance; route capture remains deferred.

Install

Requires Python 3.11+ and has no runtime dependencies (standard library only).

```
pip install evalctl
```

For unreleased changes, install from the default branch:

```
pip install "git+https://github.com/evalctl/evalctl.git"
```

Quickstart

Scaffold a project, author a suite, run it, and read the report:

```
evalctl init --json
evalctl suite add demo --runner-argv "python3 $EVALCTL_WORKSPACE/r.py" --json
evalctl case add demo --task "do X" --workspace fixtures/x --expect-json '{"exact":"ok"}'
--json
evalctl scorer add demo --name exact --required --json
evalctl run demo --json
evalctl report <run-id> --format json
```

Durable runs

Every run writes run.json before executing cases, and writes each case's state.json only after the artifacts needed for reports are complete. If a process is killed mid-run, run --resume reuses the original suite snapshot and run parameters, skips terminal cases, deletes partial unfinished case directories, and executes only the remainder.

Reports and artifact replay do not require the durability sidecars; the report hash stays based on the report projection, so runs stay comparable.

Doctor and plan

evalctl doctor reports runtime, suite root, runs root, reservations, and runner-safety state without failing just because a component is degraded.

evalctl plan resolves the case set without creating run directories, enqueueing jobs, executing runners, or scoring — a side-effect-free preview that includes run-id strategy, execution mode, per-case actions, and paste-ready follow-up commands.

Scorers and replay

Deterministic scorers run first. Command scorers call an external program that receives the case file, output file, and workspace, and emits one JSON verdict. Verdicts are captured once; reports and replay read that artifact and do not re-execute the scorer.

replay --failed selects failed or errored cases from the source run's recomputed report projection and re-runs only those against the current suite, writing a fresh partial run that names the run it was replayed from.

Learn more

- Documentation: https://evalctl.dev/docs
- Source: https://github.com/evalctl/evalctl

https://evalctl.dev